CHAPTER 26

Globalization

Exam Objectives

In this chapter you will learn to

- 053.20.1 Customize Language-Dependent Behavior for the Database and Individual Sessions
- 053.20.2 Work with Database and NLS Character Sets

The Oracle database has many capabilities grouped under the term *globalization* that will assist a DBA who must consider users of different nationalities. Globalization was known as National Language Support, or NLS, in earlier releases (you will still see the NLS acronym in several views and parameters), but globalization is more than linguistics: it is a comprehensive set of facilities for managing databases that must cover a range of languages, time zones, and cultural variations.

Globalization Requirements and Capabilities

Large database systems, and many small ones too, will usually have a user community that is distributed geographically, temporally, and linguistically. Consider a database hosted in Johannesburg, South Africa, with end users scattered throughout sub-Saharan Africa. Different users will be expecting data to be presented to them in Portuguese, French, and English, at least. They may be in three different time zones with different standards for the formats of dates and numbers. The situation becomes even more complex when the application is running in a three-tier environment: you may have a database in one location, several geographically distributed application servers, and users further distributed from the application servers.

It is possible for a lazy DBA to ignore globalization completely. Typically, such a DBA will take United States defaults for everything—and then let the programmers sort it out. But this is putting an enormous amount of work onto the programmers, and they may not wish to do it either. The result is an application that works but is detested by a portion of its users. But there is more to this than keeping people happy: there may well be financial implications too. Consider two competing e-commerce sites, both trying to sell goods all over the world. One has taken the trouble to translate everything into languages applicable to each customer; the other insists that all customers use American English. Which one is going to receive the most orders? Furthermore, dates and monetary formats can cause dreadful confusion when different countries have different standards. Such problems can be ignored or resolved programmatically, but a good DBA will attempt to resolve them through the facilities provided as standard within the database.

Character Sets

The data stored in a database must be coded into a character set. A *character set* is a defined encoding scheme for representing characters as a sequence of bits. Some products use the character sets provided by the host operating system. For example, Microsoft Word does not have its own character sets; it uses those provided by the Windows operating system. Other products provide their own character sets and are thus independent of whatever is provided by the host operating system. Oracle products fall into the latter group: they ship with their own character sets, which is one reason why Oracle applications are the same on all platforms, and why clients and servers can be on different platforms.

A character set consists of a defined number of distinct characters. The number of characters that a character set can represent is limited by the number of bits the character set uses for each character. A single-byte character set will use only one byte per character: eight bits, though some single-byte character sets restrict this even further by using only seven of the eight bits. A multibyte character set uses one, two, or even three bytes for each character. The variations here are whether the character set is fixed-width (for example, always using two bytes per character) or variable width (where some characters will be represented in one byte, other characters in two or more).

How many characters are actually needed? Well, as a bare minimum, you need upper- and lowercase letters, the digits 0 through 9, a few punctuation marks, and some special characters to mark the end of a line, or a page break, for instance. A seven-bit character set can represent a total of 128 (2^7) characters. It is simply not possible to get more than that number of different bit patterns if you have only seven bits to play with. Seven-bit character sets are just barely functional for modern computer systems, but they are usually inadequate. They provide the characters just named, but very little else. If you need to do simple things like using box drawing characters, or printing a name that includes a letter with an accent, you may find that you can't do it with a seven-bit character set. Anything more advanced, such as storing and displaying data in Arabic or Chinese script, will be totally out of the question. Unfortunately, Oracle's default character sets are seven-bit ASCII or seven-bit EBCDIC, depending on the platform: even such widely used languages as French and Spanish cannot be written correctly in these character sets. This is a historical anomaly, dating back to the days when these character sets were pretty much the only ones in use. Eight-bit character sets can represent 256 (2^8) different characters. These will typically be adequate for any Western European language–based system, though perhaps not for some Eastern European languages, and definitely not for many Asian languages. For these more complex linguistic environments, it is necessary to use a multibyte character set.

EXAM TIP The default character set is seven bit, either ASCII or EBCDIC. If you use DBCA to create a database, it will pick up a default from the operating system. This will often be better, but may not be perfect.

Unicode character sets deserve a special mention. Unicode is an international standard for character encoding, which is intended to include every character that will ever be required by any computer system. Currently, Unicode has defined more than thirty-two thousand characters.

TIP Oracle Corporation recommends AL32UTF8, a varying-width Unicode character set, for all new deployments.

Encoding Scheme	Example Character Sets
Single-byte seven-bit	US7ASCII. This is the default for Oracle on non-IBM systems. YUG7ASCII. Seven-bit Yugoslavian, a character set suitable for the languages used in much of the Balkans.
Single-byte eight-bit	WE8ISO8859P15. A Western European eight-bit ISO standard character set, which includes the Euro symbol (unlike WE8ISO8859P1). WE8DEC. Developed by Digital Equipment Corporation, widely used in the DEC (or Compaq) environment in Europe. I8EBCDIC1144. An EBCDIC character set specifically developed for Italian. EBCDIC is used on IBM platforms.
Fixed-width multibyte	AL16UTF16. This is a Unicode two-byte character set, and the only fixed-width Unicode character set supported by 11*g*.
Varying-width	JA16SJIS. Shift-JIS, a Japanese character set, where a shift-out control code is used to indicate that the following bytes are double-byte characters. A shift-in code switches back to single-byte characters. ZHT16CCDC. A traditional Chinese character set, where the most significant bit of the byte is used to indicate whether the byte is a single character or part of a multibyte character. AL32UTF8. A Unicode varying-width character set.

Table 26-1 Sample Oracle Database 11*g* Character Sets

Oracle Database 11*g* ships with more than 200 character sets. Table 26-1 includes just a few examples.

Language Support

The number of languages supported by Oracle depends on the platform, release, and patch level of the product. To determine the range available on any one installation, query the view `V$NLS_VALID_VALUES`, as follows:

```
SQL> select * from v$nls_valid_values where parameter='LANGUAGE';

PARAMETER            VALUE                ISDEP
-------------------- -------------------- -----
LANGUAGE             AMERICAN             FALSE
LANGUAGE             GERMAN               FALSE
LANGUAGE             FRENCH               FALSE
LANGUAGE             CANADIAN FRENCH      FALSE
LANGUAGE             SPANISH              FALSE
. . .
```

```
LANGUAGE             ALBANIAN             FALSE
LANGUAGE             BELARUSIAN           FALSE
LANGUAGE             IRISH                FALSE
67 rows selected.
SQL>
```

The language used will determine the language for error messages and also set defaults for date language and sort orders. The defaults are shown here:

Initialization Parameter	Default	Purpose
NLS_LANGUAGE	AMERICAN	Language for messages
NLS_DATE_LANGUAGE	AMERICAN	Used for day and month names
NLS_SORT	BINARY	Linguistic sort sequence

The default sort order—binary—is poor. Binary sorting may be acceptable for a seven-bit character set, but for character sets of eight bits or more the results are often inappropriate. For example, the ASCII value of a lowercase letter *a* is 97, and a lowercase letter *z* is 122. So a binary sort will place *a* before *z*, which is fine. But consider diacritic variations: a lowercase letter *a* with an umlaut, *ä*, is 132, which is way beyond *z*; so the binary sort order will produce "a,z,ä"—which is wrong in any language. The German sort order would give "a,ä,z"—which is correct. Figure 26-1 illustrates how a sort order is affected by the language setting, using German names.

Oracle provides many possible sort orders; there should always be one that will fit your requirements. Again, query V$NLS_VALID_VALUES to see what is available:

```
SQL> select * from v$nls_valid_values where parameter='SORT';
PARAMETER            VALUE                ISDEP
-------------------- -------------------- -----
SORT                 BINARY               FALSE
SORT                 WEST_EUROPEAN        FALSE
SORT                 XWEST_EUROPEAN       FALSE
SORT                 GERMAN               FALSE
SORT                 XGERMAN              FALSE
SORT                 DANISH               FALSE
SORT                 XDANISH              FALSE
SORT                 SPANISH              FALSE
SORT                 XSPANISH             FALSE
SORT                 GERMAN_DIN           FALSE
. . .
SORT                 SCHINESE_STROKE_M    FALSE
SORT                 GBK                  FALSE
SORT                 SCHINESE_RADICAL_M   FALSE
SORT                 JAPANESE_M           FALSE
SORT                 KOREAN_M             FALSE

87 rows selected.
```

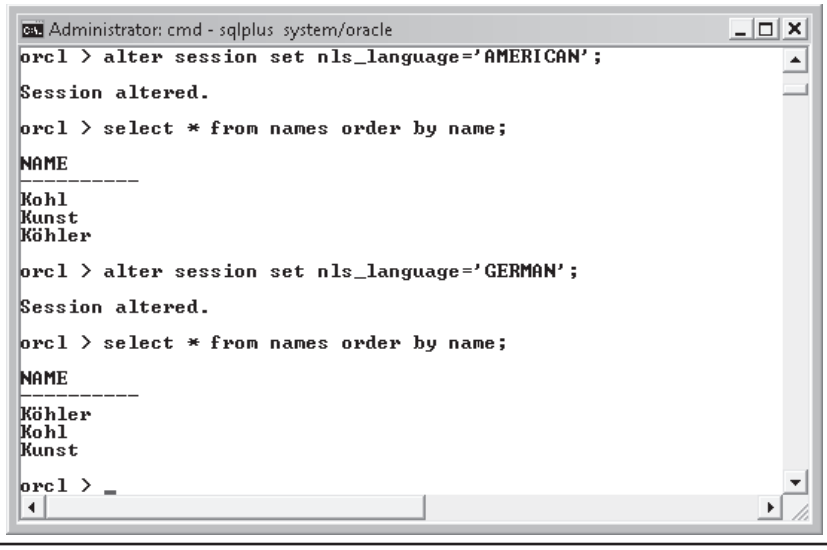


Figure 26-1 Linguistic sorting

Territory Support

The territory selected sets a number of globalization defaults. To determine the territories your database supports, again query `V$NLS_VALID_VALUES`:

```
SQL> select * from v$nls_valid_values where parameter='TERRITORY';

PARAMETER            VALUE                ISDEP
-------------------- -------------------- -----
TERRITORY            AMERICA              FALSE
TERRITORY            UNITED KINGDOM       FALSE
TERRITORY            GERMANY              FALSE
TERRITORY            FRANCE               FALSE
TERRITORY            CANADA               FALSE
TERRITORY            SPAIN                FALSE
TERRITORY            ITALY                FALSE
TERRITORY            THE NETHERLANDS      FALSE
TERRITORY            SWEDEN               FALSE
TERRITORY            NORWAY               FALSE
. . .
TERRITORY            BELARUS              FALSE

98 rows selected.
```

The territory selection sets defaults for day and week numbering, credit and debit symbols, date formats, decimal and group numeric separators, and currency symbols. Some of these can have profound effects on the way your application software will behave.

For example, in the U.S. the decimal separator is a point (.), but in Germany and many other countries it is a comma (,). Consider a number such as "10,001". Is this ten thousand and one, or ten and one thousandth? You certainly need to know. Of equal importance is day of the week numbering. In the U.S., Sunday is day 1 and Saturday is day 7, but in Germany (and indeed in most of Europe) Monday (or Montag, to take the example further) is day 1 and Sunday (Sonnabend) is day 7. If your software includes procedures that will run according to the day number, the results may be disastrous if you do not consider this. Figure 26-2 illustrates some other territory related differences in time settings.

These are the defaults for territory-related settings:

Variable	Default / Purpose
NLS_TERRITORY	AMERICA / Geographical location
NLS_CURRENCY	$ / Local currency symbol
NLS_DUAL_CURRENCY	$ / A secondary currency symbol for the territory
NLS_ISO_CURRENCY	AMERICA / Indicates the ISO territory currency symbol
NLS_DATE_FORMAT	DD-MM-RR / Format used for columns of data type DATE
NLS_NUMERIC_CHARACTERS	., / Decimal and group delimiters
NLS_TIMESTAMP_FORMAT	DD-MM-RRHH.MI.SSXFF AM / Format used for columns of data type TIMESTAMP
NLS_TIMESTAMP_TZ_FORMAT	DD-MM-RRHH.MI.SSXFF AM TZR / Format used for columns of data type TIMESTAMP WITH LOCAL TIMEZONE

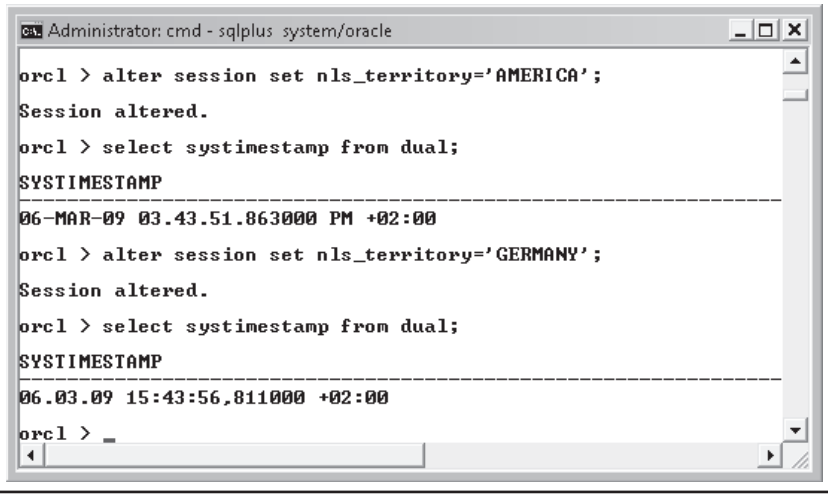


Figure 26-2 Date and time formats, on the sixth of March in the afternoon, in a time zone two hours ahead of Greenwich Mean Time (GMT)

Other NLS Settings

Apart from the language- and territory-related settings just described, there are a few more advanced settings that are less likely to cause problems:

Variable	Default / Purpose
NLS_CALENDAR	Gregorian / Allows use of alternative calendar systems
NLS_COMP	BINARY / The alternative of ANSI compares letters using their NLS value, not the numeric equivalent
NLS_LENGTH_SEMANTICS	BYTE / Allows one to manipulate multibyte characters as complete characters rather than bytes
NLS_NCHAR_CONV_EXCP	FALSE / Limits error messages generated when converting between VARCHAR2 and NVARCHAR

Figure 26-3 illustrates switching to the Japanese Imperial calendar (which counts the years from the ascension of Emperor Akihito to the throne), with an associated effect on the date display.

Using Globalization Support Features

Globalization can be specified at any and all of five levels:

- The database
- The instance
- The client environment
- The session
- The statement

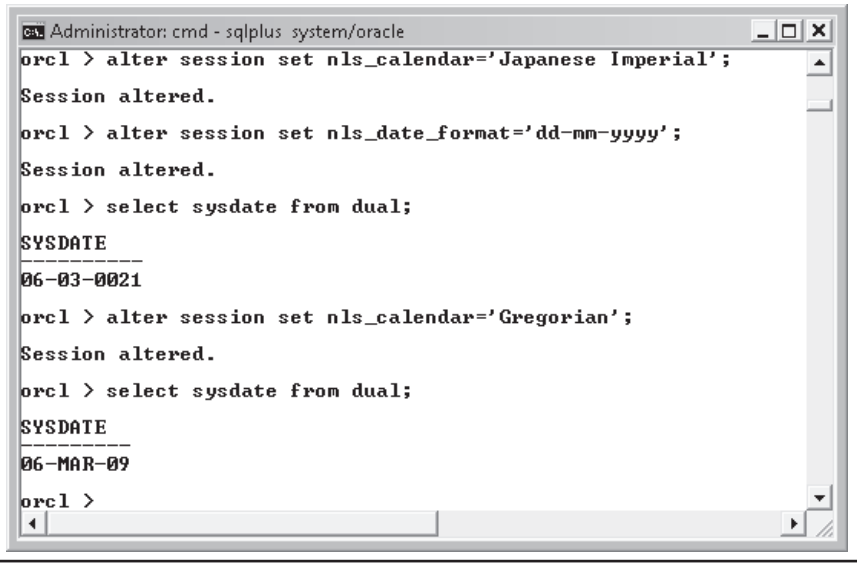


Figure 26-3 Use of the Japanese Imperial calendar

The levels are listed in ascending order of priority. Thus, instance settings take precedence over database settings, and so on. An individual statement can control its own globalization characteristics, thus overriding everything else.

EXAM TIP Remember the precedence of the various points where globalization settings can be specified. On the server side, instance settings take precedence over database settings, but all the server settings can be overridden on the client side: first by the environment, then at the session and statement levels.

Choosing a Character Set

At database creation time, choice of character set is one of the two most important decisions you make. When you create a database, two settings are vital to get right at creation time; everything else can be changed later. These two are the `DB_BLOCK_SIZE` parameter, which can never be changed, and the database character set, which it may be possible but not necessarily practicable to change. The difficulty with the `DB_BLOCK_SIZE` is that this parameter is used as the block size for the `SYSTEM` tablespace. You can't change that without recreating the data dictionary: in other words, creating a new database. The database character set is used to store all the data in columns of type `VARCHAR2`, `CLOB`, `CHAR`, and `LONG` (although still supported, you should not be using `LONG` datatypes unless you need them for backward compatibility). If you change it, you may well destroy all the data in your existing columns of these types.

It is therefore vital to select, at creation time, a character set that will fulfill all your needs, present and future. For example, if you are going to have data in French or Spanish, a Western European character set is needed. If you are going have data in Russian or Czech, you should choose an Eastern European character set. But what if you may have both Eastern and Western European languages? Furthermore, what if you anticipate a need for Korean or Thai as well? Oracle provides two solutions to the problem: the National Character Set, and the use of Unicode.

The National Character Set was introduced with release 8.0 of the database. This is a second character set, specified at database creation time, which is used for columns of data types `NVARCHAR2`, `NCLOB`, and `NCHAR`. So if the DBA anticipated that most of the information would be in English but that some would be Japanese, they could select a Western European character set for the database character set, and a Kanji character set as the National Character Set. With release 9*i*, the rules changed: from then on, the National Character Set can only be Unicode. This should not lead to any drop in functionality, because the promise of Unicode is that it can encode any character. Two types of Unicode are supported as the National Character Set: AL16UTF16 and UTF8. AL16UTF16 is a fixed-width, two-byte character set, and UTF8 is a variable-width character set. The choice between the two is a matter of space efficiency and performance, related to the type of data you anticipate storing in the `NVARCHAR2` and `NCLOB` columns.

It may very well be that the majority of the data could in fact be represented in one byte, and only a few characters would need multiple bytes. In that case, AL16UTF16 will nearly double the storage requirements—quite unnecessarily, because one of the two bytes per character will be packed with zeros. This not only wastes space but also impacts on disk I/O. UTF8 will save a lot of space. But if the majority of the data cannot be coded

in one byte, then UTF8 becomes much less efficient because the multibyte characters must be assembled, at runtime, from a number of single bytes, with a consequent performance hit. Also, UTF8 will often need three or even four bytes to store a character that AL16UTF16 can encode in two.

The second possibility for a fully multilingual database is to use Unicode as the actual database character set. The supported options are UTF8 and AL32UTF8, which are both variable-width multibyte character sets.

The only limitation on the database character set is that it must have either US7ASCII or EBCDIC as a subset. This is because the database character set is used to store SQL and PL/SQL source code, which is written in these characters.

Both the database character set and the National Character Set are specified in the `CREATE DATABASE` command. The defaults are US7ASCII and AL16UTF16. If you create a database using the Database Creation Assistant (DBCA), DBCA will provide a default for the database character set, which it will pick up from the character set of the host operating system where you are running DBCA. This may be more appropriate than the seven-bit Oracle default, but remember that your clients may be using terminals with a different operating system from the database server.

Changing Character Sets

There are many occasions when DBAs have wished that they could change the database character set. Typically, this is because the database was created using the default of US7ASCII, and later on a need arises for storing information using characters not included in that character set, such as a French name. Prior to release 9*i* there was no supported technique for changing the character set. From 9*i* onward, there is a supported technique, but there is no guarantee that it will work. It is your responsibility as DBA to carry out thorough checks that the change will not damage the data. The problem is simply that a change of character set does not reformat the data currently in the datafiles, but it will change the way the data is presented. For example, if you were to convert from a Western European character set to an Eastern European character set, many of the letters with the accents common in Western languages would then be interpreted as Cyrillic characters, with disastrous results.

There are two tools provided to assist with deciding on character set change: the Database Character Set Scanner and the Language and Character Set File Scanner. These are independently executable utilities, `csscan` and `lcsscan` on Unix, `csscan.exe` and `lcsscan.exe` on Windows.

The Database Character Set Scanner will log on to the database and make a pass through the datafiles, generating a report of possible problems. For example,

```
csscan system/systempassword full=y tochar=utf8
```

This command will connect to the database as user `SYSTEM` and scan through all the datafiles to check if conversion to UTF8 would cause any problems. A typical problem when going to UTF8 is that a character that was encoded in one byte in the original character set might require two bytes in UTF8, so the data might not fit in the column after the change. The scanner will produce a comprehensive report listing every row that will have problems with the new character set. You must then take appropriate action to fix the problems before the conversion, if possible.

TIP You must run the `csminst.sql` script to prepare the database for running the character set scanner.

The Language and Character Set File Scanner is a utility that will attempt to identify the language and character set used for a text file. It will function on plain text only; if you want to use it on, for example, a word processing document, you will have to remove all the control codes first. This scanner may be useful if you have to upload data into your database and do not know what the data is. The tool scans the file and applies a set of heuristics to make an intelligent guess about the language and character set of the data.

Having determined whether it is possible to change the character set without damage, execute the command `ALTER DATABASE CHARACTER SET` to make the change. The equivalent command to change the National Character Set is `ALTER DATABASE NATIONAL CHARACTER SET`. The only limitation with this command is that the target character set must be a superset of the original character set, but that does not guarantee that there will be no corruptions. That is the DBA's responsibility.

Globalization Within the Database

The database's globalization settings are fixed at creation time, according to the instance parameter settings in effect when the `CREATE DATABASE` command was issued and the character set was specified. They are visible in the view `NLS_DATABASE_PARAMETERS` as follows:

```
SQL> select * from nls_database_parameters;

PARAMETER                      VALUE
------------------------------ ----------------------------------------
NLS_LANGUAGE                   AMERICAN
NLS_TERRITORY                  AMERICA
NLS_CURRENCY                   $
NLS_ISO_CURRENCY               AMERICA
NLS_NUMERIC_CHARACTERS         .,
NLS_CHARACTERSET               WE8MSWIN1252
NLS_CALENDAR                   GREGORIAN
NLS_DATE_FORMAT                DD-MON-RR
NLS_DATE_LANGUAGE              AMERICAN
NLS_SORT                       BINARY
NLS_TIME_FORMAT                HH.MI.SSXFF AM
NLS_TIMESTAMP_FORMAT           DD-MON-RR HH.MI.SSXFF AM
NLS_TIME_TZ_FORMAT             HH.MI.SSXFF AM TZR
NLS_TIMESTAMP_TZ_FORMAT        DD-MON-RR HH.MI.SSXFF AM TZR
NLS_DUAL_CURRENCY              $
NLS_COMP                       BINARY
NLS_LENGTH_SEMANTICS           BYTE
NLS_NCHAR_CONV_EXCP            FALSE
NLS_NCHAR_CHARACTERSET         AL16UTF16
NLS_RDBMS_VERSION              11.1.0.6.0

20 rows selected.
```

Globalization at the Instance Level

Instance parameter settings will override the database settings. In a RAC environment, it is possible for different instances to have different settings, so that, for example, European and U.S. users could each log on to the database through an instance configured appropriately to their different needs. The settings currently in effect are exposed in the view `NLS_INSTANCE_PARAMETERS`, which has the same rows as `NLS_DATABASE_PARAMETERS` except for three rows to do with character sets and RDBMS version that do not apply to an instance.

The globalization instance parameters can be changed like any others, but as they are all static, it is necessary to restart the instance before any changes come into effect.

Client-Side Environment Settings

When an Oracle user process starts, it inspects the environment within which it is running to pick up globalization defaults. This mechanism means that it is possible for users who desire different globalization settings to configure their terminals appropriately to their needs, and then Oracle will pick up and apply the settings automatically, without the programmers or the DBA having to take any action. This feature should be used with care, as it can cause confusion because it means that the application software may be running in an environment that the programmers had not anticipated. The internal implementation of this is that the user process reads the environment variables and then generates a series of `ALTER SESSION` commands to implement them.

The key environment variable is `NLS_LANG`. The full specification for this is a language, a territory, and a character set. To use French as spoken in Canada with a Western European character set, an end user could set it to

```
NLS_LANG=FRENCH_CANADA.WEISO8859P1
```

and then, no matter what the database and instance globalization is set to, his user process will then display messages and format data according to Canadian French standards. When the user sends data to the server, he will enter it using Canadian French conventions, but the server will then store it according to the database globalization settings. The three elements (language, territory, and character set) of `NLS_LANG` are all optional.

TIP The DBA has absolutely no control over what end users do with the NLS_LANG environment variable. If the application is globalization sensitive, the developers should take this into account and control globalization within the session instead.

The conversion between server-side and client-side globalization settings is done by Oracle Net. In terms of the OSI seven-layer model, any required conversion is a layer 6 (presentation layer) function that is accomplished by Oracle Net's Two-Task Common layer. Some conversion is perfectly straightforward and should always succeed. This is the case with formatting numbers, for instance. Other conversions

are problematic. If the client and the server are using different character sets, it may not be possible for data to be converted. An extreme case would be a client process using a multibyte character set intended for an Oriental language, and a database created with US7ASCII. There is no way that the data entered on the client can be stored correctly in the much more limited character set available within the database, and data loss and corruption are inevitable.

Exercise 26-1: Make Globalization and Client Environment Settings This exercise will demonstrate how you, acting as an end user, can customize your environment, in order to affect your Oracle sessions.

1. From an operating system prompt, set the `NLS_LANG` variable to (for example) Hungarian, and also adjust the date display from the default. Using Windows,

```
C:\>set NLS_LANG=Hungarian
C:\>set NLS_DATE_FORMAT=Day dd Month yyyy
```

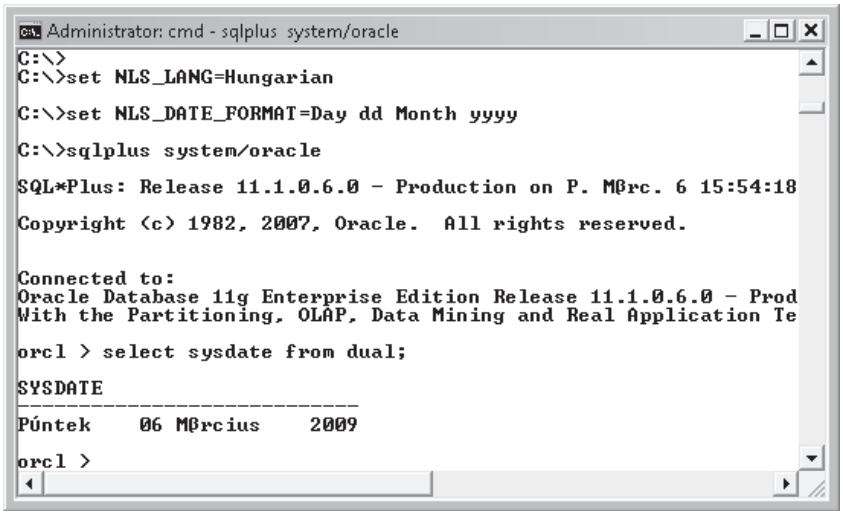
   or on Unix,

```
$ export NLS_LANG=Hungarian
$ export NLS_DATE_FORMAT='Day dd Month yyyy'
```

2. From the same operating system session, launch SQL*Plus and connect as user `SYSTEM`.
3. Display the current date with

```
select sysdate from dual;
```

The illustration shows the complete sequence of steps. Note that in the illustration the display is in fact incorrect: in Hungarian, "Friday" is "Péntek" and "March" is "Március". These errors are because the client-side settings cannot display the database character set correctly. Your date elements may differ from the illustration, depending on your server-side character set.



Session-Level Globalization Settings

Once connected, users can issue ALTER SESSION commands to set up their globalization preferences. Normally this would be done programmatically, perhaps by means of a logon trigger. The application will determine who the user is and configure the environment accordingly. An alternative to ALTER SESSION is the supplied package DBMS_SESSION. The following examples will each have the same effect:

```
SQL> alter session set nls_date_format='dd.mm.yyyy';
Session altered.
SQL> execute dbms_session.set_nls('nls_date_format','''dd.mm.yyyy''');
PL/SQL procedure successfully completed.
```

Specifications at the session level take precedence over the server-side database and instance settings and will also override any attempt made by the user to configure their session with operating system environment variables. The globalization settings currently in effect for your session are shown in the V$NLS_PARAMETERS view. The same information, with the exception of the character sets, is shown in the NLS_SESSION_PARAMETERS view.

Exercise 26-2: Control Globalization Within the Session For this exercise, it is assumed that you have completed Exercise 26-1 and that you are working in the same SQL*Plus session. You will demonstrate how European and U.S. standards can cause confusion.

1. Confirm that your NLS_LANG environment variable is set to a European language. On Windows,

   ```
   SQL> host echo %NLS_LANG%
   ```

   or on Unix,

   ```
   SQL> host echo $NLS_LANG
   ```

2. Set your date display to show the day number:

   ```
   SQL> alter session set nls_date_format='D';
   ```

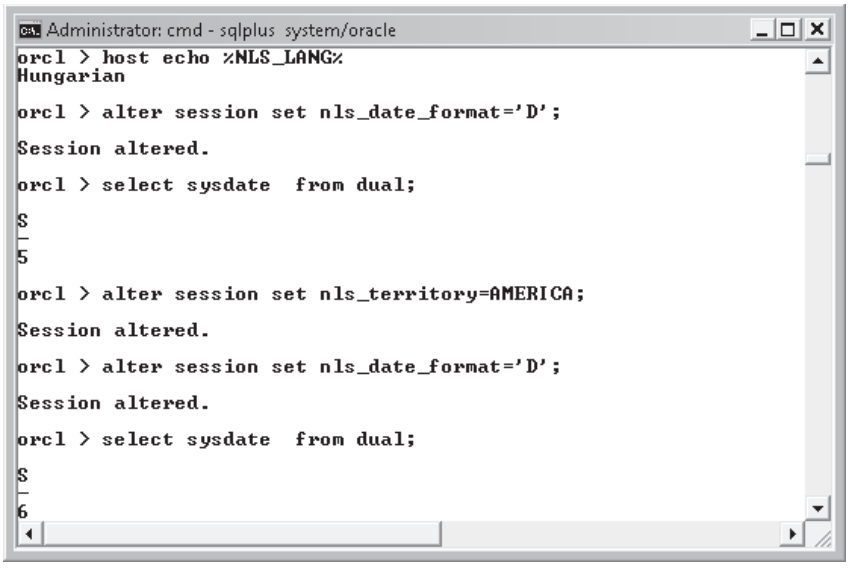
3. Display the number of today's day:

   ```
   SQL> select sysdate from dual;
   ```

4. Change your territory to the U.S., and again set the date display format:

   ```
   SQL> alter session set nls_territory=AMERICA;
   SQL> alter session set nls_date_format='D';
   ```

5. Issue the query from Step 3 again, and note that the day number has changed with the shift of environment from Europe to America as shown in the following illustration:



Statement Globalization Settings

The tightest level of control over globalization is to manage it programmatically, within each SQL statement. This entails using NLS parameters in SQL functions. Figure 26-4 shows an example that presents the same data in two date languages.

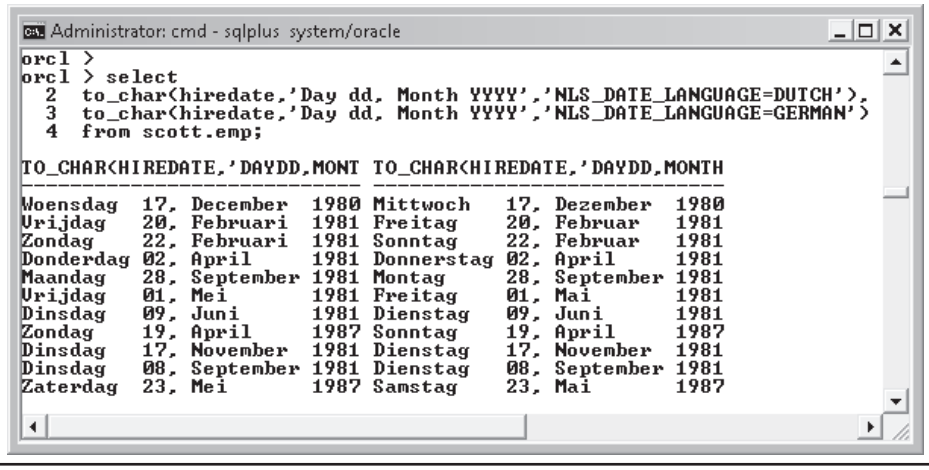


Figure 26-4 Controlling date language within a SQL statement

The SQL functions to consider are the typecasting functions that convert between data types. Depending on the function, various parameters may be used.

Function	Globalization Parameters
TO_DATE	NLS_DATE_LANGUAGE
	NLS_CALENDAR
TO_NUMBER	NLS_NUMERIC_CHARACTERS
	NLS_CURRENCY
	NLS_DUAL_CURRENCY
	NLS_ISO_CURRENCY
	NLS_CALENDAR
TO_CHAR, TO_NCHAR	NLS_DATE_LANGUAGE
	NLS_NUMERIC_CHARACTERS
	NLS_CURRENCY
	NLS_DUAL_CURRENCY
	NLS_ISO_CURRENCY
	NLS_CALENDAR

Numbers, dates, and times can have a wide range of format masks applied for display. Within numbers, these masks allow embedding group and decimal separators, and the various currency symbols; dates can be formatted as virtually any combination of text and numbers; times can be shown with or without time zone indicators and as AM/PM or twenty-four hours. Refer to Chapter 10 for a discussion of conversion functions and format masks.

Languages and Time Zones

Once you have your NLS settings in place, you need to understand how they are used when sorting or searching. Depending on the language, the results of a sort on a name or address in the database will return the results in a different order.

Even with Oracle's robust support for character sets, there are occasions when you might want to create a customized globalization environment for a database, or tweak an existing locale. In a later section, a brief introduction to the Oracle Locale Builder is provided.

The chapter concludes with a discussion of time zones, and how Oracle supports them using initialization parameters at both the session and database levels, much like NLS parameters.

Linguistic Sorting and Selection

Oracle's default sort order is binary. The strings to be sorted are read from left to right, and each character is reduced to its numeric ASCII (or EBCDIC) value. The sort is done in one pass. This may be suitable for American English, but it may give incorrect results for other languages. Obvious problems are diacritics such as *ä* or *à* and diphthongs like *æ*, but there are also more subtle matters. For example, in traditional Spanish, *ch* is a character in its own right that comes after *c*; thus the correct order is "Cerveze, Cordoba, Chavez." To sort this correctly, the database must inspect the subsequent character as well as the current character, if it is a *c*.

TIP As a general rule, it is safe to assume that Oracle can handle just about any linguistic problem, but that you as DBA may not be competent to understand it. You will need an expert in whatever languages you are working in to advise.

Linguistic sorting means that rather than replacing each character with its numeric equivalent, Oracle will replace each character with a numeric value that reflects its correct position in the sequence appropriate to the language in use. There are some variations here, depending on the complexity of the environment.

A monolingual sort makes two passes through the strings being compared. The first pass is based on the *major* value of each character. The major value is derived by removing diacritic and case differences. In effect, each letter is considered as uppercase with no accents. Then a second comparison is made, using the *minor* values, which are case and diacritic sensitive. Monolingual sorts are much better than binary but are still not always adequate. For French, for example, Oracle provides the monolingual FRENCH sort order, and the multilingual FRENCH_M, which may be better if the data is not exclusively French.

A technique that may remove confusion is to use Oracle's case- and diacritic-insensitive sort options. For example, you may wish to consider these variations on a Scottish name as equivalent:

MacKay
Mackay
MACKAY

To retrieve all three with one query, first set the `NLS_SORT` parameter to `GENERIC_BASELETTER` as shown in Figure 26-5. This will ignore case and diacritic variations. Then set the `NLS_COMP` parameter away from the default of `BINARY` to `ANSI`. This instructs Oracle to compare values using the `NLS_SORT` rules, not the numeric value of the character. The `GENERIC_BASELETTER` sort order will also "correct" what may appear to some as incorrect ordering. A more complex example would require equating "McKay" with "MacKay"; that would require the Locale Builder.

Similarly, all the sort orders can be suffixed with _AI or _CI for accent-insensitive and case-insensitive sorting. For example,

```
SQL> alter session set nls_sort=FRENCH_CI;
```

will ignore upper- and lowercase variations but will still handle accented characters according to French standards.

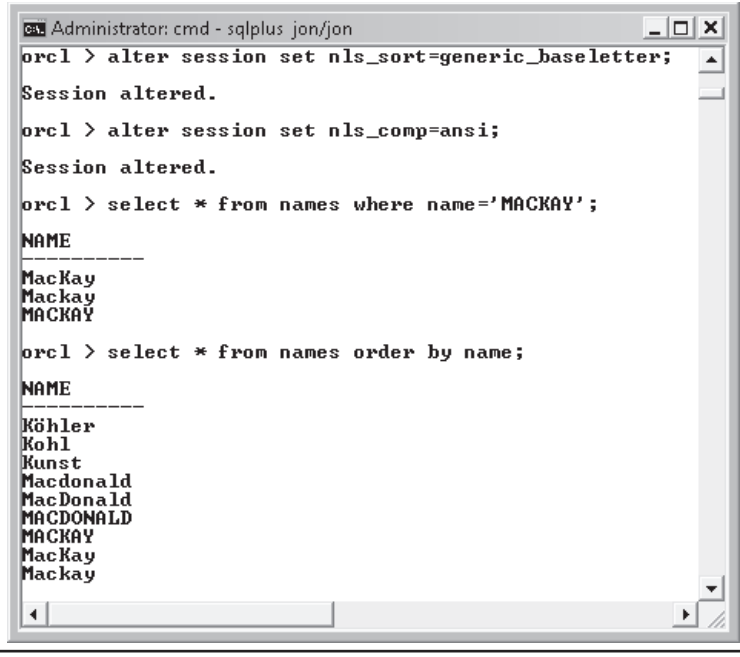


Figure 26-5 Case and accent insensitivity for SELECT and sorting

The Locale Builder

The globalization support provided as standard by Oracle Database 11*g* is phenomenal, but there may be circumstances that it cannot handle. The Locale Builder is a graphical tool that can create a customized globalization environment, by generating definitions for languages, territories, character sets, and linguistic sorting.

As an example, Oracle does not provide out-of-the-box support for Afrikaans; you could create a customized globalization to fill this gap, which might combine elements of Dutch and English standards with customizations common in Southern Africa such as ignoring the punctuation marks or spaces in names like O'Hara or Du Toit. To launch the Locale Builder, run

```
$ORACLE_HOME/nls/lbuilder/lbuilder
```

on Unix, or

```
%ORACLE_HOME%\nls\lbuilder\lbuilder.bat
```

on Windows to view the dialog box shown in Figure 26-6.

Using Time Zones

Businesses, and therefore databases, must work across time zones. From release 9*i* onward, the Oracle environment can be made time zone aware. This is done by specifying a time zone in which the database operates, and then using the `TIMESTAMP`

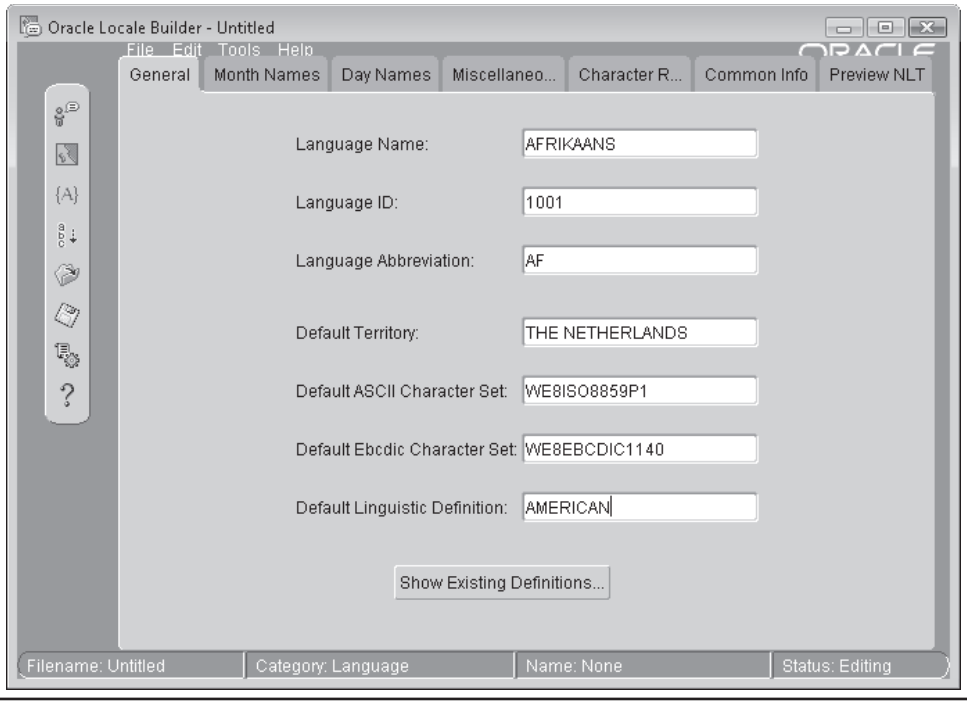


Figure 26-6 Creating a locale with the Locale Builder

`WITH TIME ZONE` and `TIMESTAMP WITH LOCAL TIME ZONE` data types. The former will be not be normalized to the database time zone when it is stored, but it will have a time zone indicator to show the zone to which it refers. The latter is normalized to the database time zone on storage but is subsequently converted to the client time zone on retrieval. The usual `DATE` and `TIMESTAMP` data types are always normalized to the database time zone on storage and displayed unchanged when selected.

As an example of when time zone processing is important, consider an e-mail database hosted in London, set to Greenwich Mean Time, GMT. A user in Harare (which is two hours ahead of GMT) sends an e-mail at his local time of 15:00; the mail is addressed to two recipients, one in Paris (Central European Time, CET: one hour ahead of GMT with daylight saving time in effect in the Northern hemisphere summer) and the other in Bogotá (which is five hours behind GMT). How do you ensure that the recipients and the sender will all see the mail as having been sent correctly according their local time zone? If the column denoting when the mail was sent is of data type `TIMESTAMP WITH LOCAL TIME ZONE`, then when the mail is received by the database, the time will be normalized to GMT: it will be saved as 13:00. Then when the Bogotá user retrieves it, the time will be adjusted to 08:00 by his user process. When the Paris user retrieves the mail, they will see it as having been sent at either 14:00 or 15:00, depending on whether the date it was sent was in the period between March and October when daylight saving time is in effect. It is possible to do this type of work programmatically, but it requires a great deal of work

as well as knowledge of all time zones and any local quirks for daylight saving. The database can do it all for you.

The database time zone can be set at creation time in the CREATE DATABASE command and adjusted later with ALTER DATABASE SET TIME_ZONE=. If not set, it defaults to the time zone picked up from the host operating system at the time of creation. The client time zone defaults to that of the client operating system, or it can be set with the environment variable ORA_STDZ. Within a session, the time zone can be set with ALTER SESSION SET TIME_ZONE=. Time zones can always be specified by full name, by abbreviated name, or as a fixed offset, in hours and minutes, from GMT. The last option cannot take account of daylight saving time adjustments. The list of supported time zones is displayed in V$TIMEZONE_NAMES.

Exercise 26-3: Make Time Zone Adjustments Confirm and adjust your current time zone, using appropriate data types. Test the results using appropriate formatting masks.

1. Using SQL*Plus, connect to your instance as user SYSTEM.
2. Identify the database time zone with this query:

```
select property_value from database_properties
where property_name = 'DBTIMEZONE';
```

   and note the result.

3. Create a table as follows:

```
create table times
     (date_std date,
      date_tz timestamp with time zone,
      date_ltz timestamp with local time zone);
```

4. View the list of supported time zones with this query:

```
select * from v$timezone_names;
```

5. Adjust your session time zone to something other than the database time zone, for example,

```
alter session set time_zone='Pacific/Tahiti';
```

6. Set your timestamp with time zone format to twenty-four-hour clock, with abbreviated time zone names with daylight saving variations.

```
alter session set nls_timestamp_tz_format='YYYY-MM-DD HH24:MI:SS TZD';
```

7. Set your timestamp format to twenty-four-hour clock.

```
alter session set nls_timestamp_format='YYYY-MM-DD HH24:MI:SS';
```

8. Set your date format to twenty-four-hour clock.

```
alter session set nls_date_format='YYYY-MM-DD HH24:MI:SS';
```

9. Insert a row into the table created in Step 3.

```
insert into times values('2008-10-26 15:00:00',
'2008-10-26 15:00:00','2008-10-26 15:00:00');
```

10. Display the times.

```
select * from times;
```

    Note that all times read 15:00.

11. Switch your session to the database time zone.

```
alter session set time_zone=DBTIMEZONE;
```

12. Repeat the query from Step 10, and note that the `TIMESTAMP WITH LOCAL TIMEZONE` has been adjusted to reflect that your session is now in a different zone.

13. Tidy up:

```
drop table times;
```


Two-Minute Drill

Customize Language-Dependent Behavior for the Database and Individual Sessions

- Globalization covers aspects of data presentation, calendars, dates, including dates, numbers and linguistic localizations.
- A *character set* is a defined encoding scheme for representing characters as a sequence of bits.
- The number of characters that a character set can represent is limited by the number of bits the character set uses for each character.
- The Unicode standards are an international standard for character encoding, which is intended to include every character that will ever be required by any computer system.
- The number of languages supported by Oracle depends on the platform, release, and patch level of the product.
- The language used will determine the language for error messages and also set defaults for date language and sort orders.
- Binary sorting may be acceptable for a seven-bit character set, but for character sets of eight bits or more the results are often inappropriate.
- Query `V$NLS_VALID_VALUES` to see the available character sets, sort orders, territories, and languages.
- Globalization can be specified at any and all of these five levels, in increasing order of priority: database, instance, client environment, session, and statement.
- The database character set is used to store all the data in columns of type `VARCHAR2`, `CLOB`, `CHAR`, and `LONG`.
- Two types of Unicode are supported as the National Character Set: AL16UTF16 and UTF8.

- There are two tools provided to assist with deciding on character set change: the Database Character Set Scanner and the Language and Character Set File Scanner.
- The key client-side environment variable is `NLS_LANG`. The full specification for this is a language, a territory, and a character set.

Work with Database and NLS Character Sets

- Oracle's default sort order is binary.
- Linguistic sorting means that rather than replacing each character with its numeric equivalent, Oracle will replace each character with a numeric value that reflects its correct position in the sequence appropriate to the language in use.
- The Locale Builder is a graphical tool that can create a customized globalization environment, by generating definitions for languages, territories, character sets, and linguistic sorting.
- Applications are made time-zone aware by specifying a time zone in which the database operates, and then using the `TIMESTAMP WITH TIME ZONE` and `TIMESTAMP WITH LOCAL TIME ZONE` data types.
- The usual `DATE` and `TIMESTAMP` data types are always normalized to the database time zone on storage and displayed unchanged when selected.
- The database time zone can be set at creation time in the `CREATE DATABASE` command and adjusted later with `ALTER DATABASE SET TIME_ZONE`.

Self Test

1. Your database was created with US7ASCII as the database character set, and you later find that this is inadequate. What can you do? (Choose the best answer.)
   - **A.** Recreate the database.
   - **B.** Issue an `alter database character set...` command.
   - **C.** Issue an `alter system character set...` command.
   - **D.** Generate a `create controlfile...` command, edit it to specify a different character set, and recreate the controlfile.
2. What are the options for the National Character Set? (Choose the best answer.)
   - **A.** None. It must be AL16UTF16.
   - **B.** It can be any Unicode character set.
   - **C.** It can be either AL16UTF16 or UTF8.
   - **D.** It can be any character set you require.
3. Match each character set with a type:

Character Set	Type
1. AL16UTF16	a. Seven-bit single-byte
2. US7ASCII	b. Eight-bit single-byte
3. UTF8	c. Fixed-width multibyte
4. WE8ISO8859P15	d. Variable-width

A. 1-c; 2-b; 3-d; 4-a

B. 1-d; 2-a; 3-c; 4-b

C. 1-c; 2-d; 3-b; 4-a

D. 1-c; 2-a; 3-d; 4-b

4. Which statements are correct about the TIMESTAMP WITH LOCAL TIME ZONE data type? (Choose two answers.)

   A. Data is saved with a local time zone indicator.

   B. Data is normalized to the database time zone when it is saved.

   C. On retrieval, data is normalized to the retrieving client's time zone.

   D. On retrieval, data is normalized to the time zone of the client that entered it.

5. Globalization can be set at various levels. Put these in order of precedence, lowest first:

   A. Client environment

   B. Database settings

   C. Instance parameters

   D. Session parameters

   E. Statements

6. The `NLS_LANGUAGE` and `NLS_TERRITORY` parameters set defaults for a number of other globalization parameters. Which of the following are controlled by `NLS_LANGUAGE`? (Choose two answers.)

   A. `NLS_DATE_LANGUAGE`

   B. `NLS_DATE_FORMAT`

   C. `NLS_NUMERIC_CHARACTERS`

   D. `NLS_SORT`

7. Choose the best description of the Character Set Scanner tool:

   A. It scans character sets to assess their suitability for a particular language.

   B. It scans files to determine the language and character set of the data in them.

   C. It scans datafiles to determine whether the character set can be changed.

   D. It reports on problems a character set change would cause.

8. If the database and the user process are using different character sets, how does data get converted? (Choose the best answer.)
   - **A.** Data is not converted, which is why there may be corruptions if the character sets are incompatible.
   - **B.** On data entry, the instance converts data to the database character set. On retrieval, the user process converts to the client character set.
   - **C.** Oracle Net will convert, in both directions.
   - **D.** It depends on various NLS parameters.
9. The database is set to GMT. A client in Buenos Aires (three hours behind GMT) executes these statements at 10:00:00 local time:

```
create table times(c1 timestamp,
c2 timestamp with local time zone);
insert into times values(to_timestamp('10:00:00'),
  to_timestamp('10:00:00'));
commit;
```

   A client in Nairobi (three hours ahead of GMT) executes these statements at 18:00:00 local time:

```
alter session set nls_timestamp_format='hh24:mi:ss';
select * from times;
```

   What will the Nairobi user see for the columns c1 and c2? (Choose the best answer.)
   - **A.** 10:00:00 and 16:00:00
   - **B.** 13:00:00 and 16:00:00
   - **C.** 13:00:00 and 10:00:00
   - **D.** 10:00:00 and 13:00:00
10. Study the result of this query:

```
SQL> select * from dates;
C1
--------
06-04-08
```

    C1 is a date-type column. How could you determine what the date returned actually means? (Choose three answers.)
    - **A.** Query `NLS_DATABASE_PARAMETERS`.
    - **B.** Query `NLS_INSTANCE_PARAMETERS`.
    - **C.** Query `NLS_SESSION_PARAMETERS`.
    - **D.** Set your `NLS_DATE_FORMAT` to a known value, and rerun the query.
    - **E.** Change the query to use `TO_CHAR` with an NLS parameter.
11. How can you prevent users from causing confusion with, for instance, date and time formats by setting local globalization environment variables? (Choose the best answer.)

A. You can't; the users have control over this.

B. Write logon triggers to set the session environment.

C. Set instance globalization parameters to override client-side settings.

D. Configure Oracle Net to convert all data sent to and from the database appropriately.

12. Which view will tell you what languages can be supported by your installation? (Choose the best answer.)

A. `NLS_DATABASE_PARAMETERS`

B. `NLS_INSTANCE_PARAMETERS`

C. `V$NLS_VALID_VALUES`

D. `V$NLS_LANGUAGES`

13. You want to make the order in which sorted names are returned independent of whether the names include accented characters, upper- and lowercase characters, punctuation marks, or spaces. How can you do this? (Choose the best answer.)

A. Set the sort order to `GENERIC_BASELETTER`, which will ignore such variations.

B. Use the _AI and _CI versions of any of the supported sort orders.

C. Use the Locale Builder to design a custom sort order.

D. This cannot be done.

Self Test Answers

1. ☑ **B**. Use this command, but test with the character set scanner first.

☒ **A**, **C**, and **D**. **A** is wrong because you do not need to recreate the database to change the database character set. **C** is wrong because `ALTER SYSTEM` cannot be used to change the character set. **D** is wrong because changing the character set in the control file will not convert the database character set.

2. ☑ **C**. Either of these Unicode sets is currently allowed.

☒ All other answers are wrong because the only two options are AL16UTF16 or UTF8.

3. ☑ **D**. **1-c; 2-a; 3-d; 4-b**

☒ **A**, **B**, and **C**. All other combinations are incorrect.

4. ☑ **B** and **C**. This is the data type that fully normalizes times to and from the database.

☒ **A** and **D**. Timestamp values are not saved with the time zone indicator, nor are they normalized when retrieved.

5. ☑ **B**, **C**, **A**, **D**, and **E**. The correct order is **B**, **C**, **A**, **D**, **E**. Instance parameters override the database parameters, and then on the client side environment variables can be overridden by `ALTER SESSION` commands, and then by individual statements.

   ☒ All other orders are incorrect.

6. ☑ **A** and **D**. `NLS_DATE_LANGUAGE` and `NLS_SORT` are the two parameters controlled by the `NLS_LANGUAGE`.

   ☒ **B** and **C**. `NLS_DATE_FORMAT` and `NLS_NUMERIC_CHARACTERS` are controlled by `NLS_TERRITORY`.

7. ☑ **D**. It will, for instance, report if a changed encoding would prevent data from fitting into an existing column.

   ☒ **A**, **B**, and **C**. All other options are incorrect descriptions.

8. ☑ **C**. Oracle Net will do the best conversion possible.

   ☒ **A**, **B**, and **D**. All other conversion scenarios are incorrect.

9. ☑ **B**. The database will normalize the time 10:00:00 from the local time zone at the point of entry, GMT+3, to the database time zone, GMT. Thus both times are saved as 13:00:00 GMT. For retrieval, the timestamp column will be displayed as saved, 13:00:00, but the timestamp with local time zone column will adjust the time to that of the time zone of the client retrieving the data, which is GMT+3.

   ☒ **A**, **C**, and **D**. All other options are incorrect.

10. ☑ **C, D** and **E**. `NLS_SESSION_PARAMETERS` will show the format used so that you can interpret the output of the query correctly, or you could set the format to a sensible value, or control the format in the query.

    ☒ **A**, and **D**. You must query the session-specific version of the view to be sure of interpreting the output correctly.

11. ☑ **B**. The best option is to write logon triggers, which will prevent any possible confusion caused by the client configuration.

    ☒ **A**, **C**, and **D**. **A** is wrong because you can override the local settings with a logon trigger. **C** is wrong because client-side settings can override instance settings. **D** is wrong because you cannot configure Oracle Net to perform a specific conversion.

12. ☑ **C**. The view `V$NLS_VALID_VALUES` will show you the full range of supported languages, as well as all other globalization options.

    ☒ **A**, **B**, and **D**. **A** is wrong because `NLS_DATABASE_PARAMETERS` shows the permanent values for each database NLS-related initialization parameter. **B** is wrong because NLS_INSTANCE_PARAMETERS shows the changed NLS values since instance startup. **D** is wrong because there is no such view as `V$NLS_LANGUAGES`.

13. ☑ **C**. To remove punctuation marks and spaces as well, you will need to create your own variation with the Locale Builder.

    ☒ **A**, **B**, and **D**. Setting the sort order to `GENERIC_BASELETTER` or using the `_AI` or `_CI` versions of the sort orders does not remove punctuation marks and spaces.